

1. Implementación digital

1.1 Teoría básica

En este documento vamos a presentar la forma de implementar digitalmente un controlador, partiendo de un diseño realizado analógicamente.

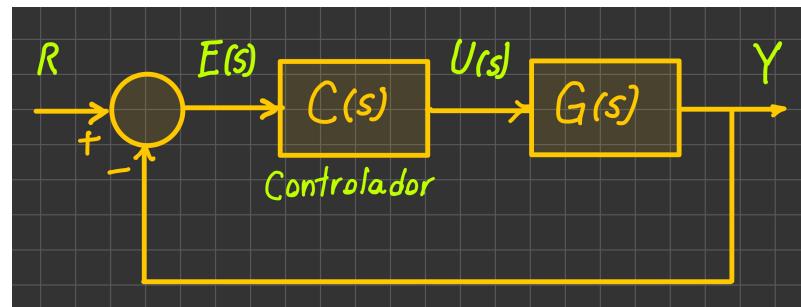


Figura 1.1: Sistema de lazo cerrado con un controlador continuo $C(s)$

Consideremos el sistema realimentado presentado en la figura 1.1. La función de transferencia de lazo cerrado, denotada $T(s)$, está dada por

$$T(s) = \frac{Y(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s)}. \quad (1.1)$$

Asumamos que para este sistema se ha diseñado un controlador analógico, denotado $C(s)$, usando algún método de síntesis de controladores. Asumamos, también, que la función de transferencia $C(s)$ es propia (es decir que el grado del numerador es menor o igual que el grado del denominador) y que no tiene polos en el semiplano derecho, aunque puede tener polos en el eje imaginario.

Con estas condiciones podemos obtener un controlador de tiempo discreto $C_d(z)$ (y su correspondiente algoritmo de procesamiento digital), el cual aproxima el comportamiento del controlador analógico $C(s)$. Los pasos para hacerlo se discuten a continuación.

1.2 Pasos para la conversión de $C(s)$ a $C_d(z)$

La figura 1.2 describe sucintamente la secuencia de pasos que debemos realizar para implementar un controlador digital. Estos pasos se describen en detalle a continuación.

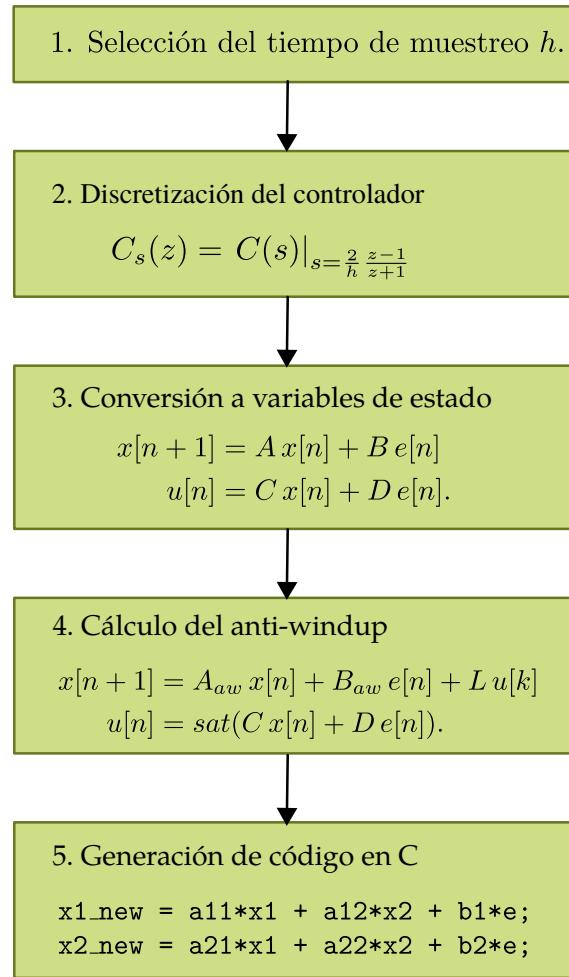


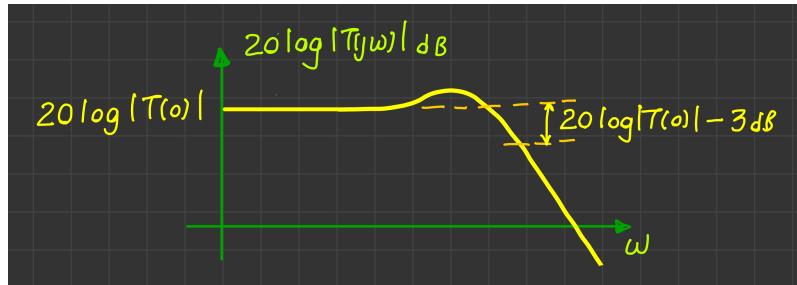
Figura 1.2: Pasos en la implementación de un controlador digital.

Paso 1: Selección del tiempo de muestreo

La realización de un controlador digital implica la selección del tiempo de muestreo usado para correr el algoritmo de control. Denotemos por h a este tiempo de muestreo y por $f_h = 1/h$ a la frecuencia de muestreo del controlador digital.

Conforme se muestra en la figura 1.3, definamos como f_b a la frecuencia de corte del sistema de lazo cerrado, esto es, la frecuencia a la cual la ganancia del sistema de lazo cerrado es:

$$|T(j\omega_b)| = 0.707 |T(0)|.$$

Figura 1.3: Ancho de banda del sistema de lazo cerrado $T(s)$.

O en decibeles:

$$20 \log |T(j\omega_b)| = 20 \log |T(0)| - 3 \text{ dB}.$$

Siendo $\omega_b = 2\pi f_b$ la frecuencia angular del sistema.

Con estas definiciones, una selección apropiada de la frecuencia de muestreo $f_h = 1/h$ para un sistema de control digital debe satisfacer la siguiente desigualdad [2]:

$$f_h \geq 10 \times f_b. \quad (1.2)$$

Así, la frecuencia de muestreo debe ser *al menos 10 veces mayor al ancho de banda del sistema retroalimentado $T(s)$* . Entre más alta sea la frecuencia de muestreo f_h (esto es, más rápido el tiempo de muestreo h), el funcionamiento del controlador digital $C_d(z)$ es más parecido a su contraparte analógica $C(s)$.

Invirtiendo la desigualdad (1.2), obtenemos la siguiente desigualdad para el tiempo de muestreo h :

$$h \leq \frac{1}{10 \times f_b}. \quad (1.3)$$

Paso 2: Discretización de controlador

Para discretizar el controlador se usa preferentemente la transformación bilineal (llamada también *aproximación de Tustin*), definida por:

$$s = \frac{2z-1}{hz+1}. \quad (1.4)$$

Mediante esta transformación aproximamos el controlador analógico $C(s)$ por el controlador digital $C_d(z)$, usando la siguiente fórmula.

$$C_d(z) = C(s)|_{s=\frac{2z-1}{hz+1}}. \quad (1.5)$$

Paso 3: Conversión a variables de estado

Una vez obtenido el controlador en el mundo discreto $C_d(z) = E(z)/U(z)$, debemos obtener una realización en variables de estado del controlador discretizado. En esta descripción, el controlador está representado de la siguiente forma:

$$x[n+1] = Ax[n] + Be[n] \quad (1.6)$$

$$u[n] = Cx[n] + De[n]. \quad (1.7)$$

Notemos que la entrada del controlador en variables de estado es el error actual $e[n]$ y la salida es la señal de control actual $u[n]$.

Es importante comentar que minimizar errores numéricos debemos usar una buena realización en variables de estado tal como la *realización modal* [1].

Paso 4: Control Anti-windup

En todo sistema de control existe una limitación energética para la señal de control $u(t)$ que un actuador real provee. Esta limitación se puede caracterizar por la siguiente función de saturación:

$$sat(u) = \begin{cases} u_{max}, & \text{si } u > u_{max} \\ u_{min}, & \text{si } u < u_{min} \\ u, & \text{en otro caso.} \end{cases}$$

Siendo u_{min} y u_{max} , respectivamente, el valor mínimo y máximo de acción de control que puede ser realmente provista por el actuador al sistema controlado.

Al igual que cuando implementamos digitalmente un controlador PID, debemos limitar el crecimiento de las variables de estado del controlador que se produce como consecuencia de la saturación del actuador causada por cambios grandes en la señal de referencia. Esto se realiza usando un vector de ganancia¹ que controla el crecimiento del vector de estado $x[n]$ definido en las ecuación (1.6).

Definamos el vector de ganancia L del control *anti-windup* como:

$$L = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_m \end{bmatrix}$$

en que m representa el orden del controlador $C(s)$, esto es, el grado del su denominador.

Multiplicando ambos lados de la ecuación (1.7) por L , obtenemos:

$$Lu[n] = LCx[n] + LD e[n] \quad (1.8)$$

Sumando, ahora, las ecuaciones (1.8) y (1.6), obtenemos:

$$\begin{aligned} x[n+1] &= Ax[n] + Be[n] + L(u[n] - Cx[n] - De[n]) \\ &= (A - LC)x[n] + (B - LD)e[n] + Lu[n]. \end{aligned} \quad (1.9)$$

¹Este vector de denominado observador

Para el cálculo del vector de ganancia L se puede usar el algoritmo del filtro de *Kalman*, incluido en la mayoría de programas de cómputo para análisis y diseño de sistemas dinámicos lineales.

Paso 5: Implementación del algoritmo de control.

Definiendo las matrices $A_{aw} = A - LC$ y $B_{aw} = B - LD$ en la ecuación (1.9) obtenemos, finalmente, las siguientes ecuaciones en diferencias que forman el algoritmo del controlador digital $C_d(z)$.

Resultado 1.2.1 — Algoritmo del controlador digital con anti-windup. Las siguientes ecuaciones son la base del código de implementación de un controlador general digital $C_d(z)$, resultante de la traducción de un controlador diseñado analógicamente $C(s)$.

$$x[n+1] = A_{aw}x[n] + B_{aw}e[n] + Lu[k] \quad (1.10)$$

$$u[n] = \text{sat}(Cx[n] + De[n]). \quad (1.11)$$

Notemos que las ecuaciones (1.10) y (1.11) contienen, en sí mismas, la forma en que debe programarse un controlador en un lenguaje como C. Esto lo ilustramos en el siguiente ejemplo, el cual también se desarrolla en detalle en el notebook adjunto a este documento.

1.3 Ejemplo desarrollado

En este ejemplo vamos a obtener el código que implementa un controlador digital $C_d(z)$, resultante de la traducción de un diseño analógico.

Consideremos el control de ángulo del motor del laboratorio, definido por la siguiente función de transferencia:

$$G(s) = \frac{1232.9}{s(0.33s + 1)}.$$

Usando algún método de síntesis, hemos obtenido el siguiente controlador:

$$C(s) = \frac{0.05506(s + 26)(s + 3.055)}{s^2 + 40.72s + 613.4}. \quad (1.12)$$

Paso 1: Selección del tiempo de muestreo

Mediante simulación encontramos el ancho de banda del sistema de lazo cerrado. De acuerdo con la figura 1.4, el ancho de banda del sistema de lazo cerrado es $\omega_b = 12.63$. De donde $f_b = \omega_b/(2\pi) = 2.01$ Hz.

Con base en este resultado, la frecuencia de muestreo f_h debe ser, al menos:

$$\begin{aligned} f_h &\geq 10 \times f_b \\ &\geq 20.1 \text{Hz} \end{aligned}$$

Es decir el tiempo de muestreo h debe ser menor o igual a $1/f_h$, esto es:

$$h \leq \frac{1}{20.1} \approx 50 \text{ms}. \quad (1.13)$$

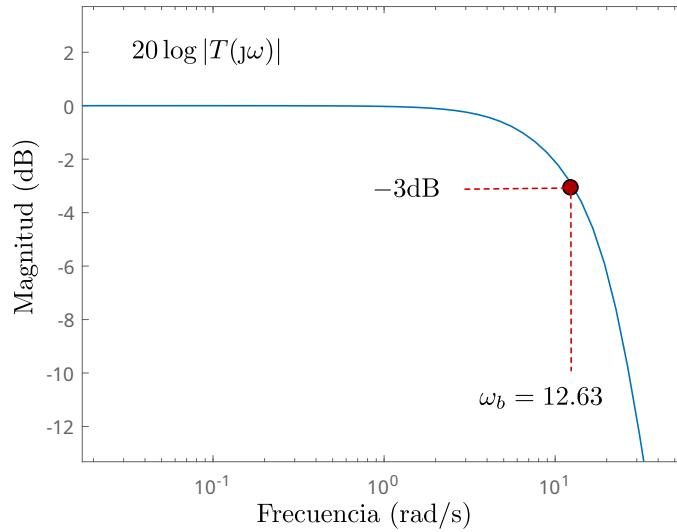


Figura 1.4: Ancho de banda del sistema de lazo cerrado $T(s)$.

Entonces, el tiempo de muestreo h debe ser de, como mínimo, 50 ms. Seleccionaremos $h = 10$ ms.

Paso 2: Discretización del controlador

Obtenemos el controlador discreto $C_d(s)$ usando la transformación bilineal.

$$\begin{aligned} C_d(z) &= C(s)\Big|_{s=\frac{2}{h}\frac{z-1}{z+1}} \\ &= \frac{0.05506 \left(\frac{2}{0.01} \frac{z-1}{z+1} + 26 \right) \left(\frac{2}{0.01} \frac{z-1}{z+1} + 3.055 \right)}{\left(\frac{2}{0.01} \frac{z-1}{z+1} \right)^2 + 40.72 \left(\frac{2}{0.01} \frac{z-1}{z+1} \right) + 613.4} \end{aligned}$$

Usando la función `c2d(Controller, h, 'tustin')` obtenemos directamente $C_d(z)$:

$$C_d(z) = \frac{0.05182z^2 - 0.09016z + 0.0387}{z^2 - 1.616z + 0.6659}$$

Paso 3: Conversión a variables de estado

Obtenemos la descripción en variables de estado del controlador $C_d(z)$ de la ecuación 1.3. Como $C_d(z)$ es de grado 2 su realización en variables de estado tiene la siguiente forma:

$$\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_A \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}}_B e[n] \quad (1.14)$$

$$u[n] = \underbrace{\begin{bmatrix} c_1 & c_2 \end{bmatrix}}_C \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + D e[n] \quad (1.15)$$

Es conveniente usar una realización llamada *modal* por sus buenas características numéricas [1]. Esto lo hacemos en Julia usando la función `modal_form(Controller)` o la función `modalreal(Controller)` en MATLAB. Así entonces, obtenemos los siguientes valores numéricos para las ecuaciones (1.14) y (1.15):

$$\begin{aligned} \begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix} &= \underbrace{\begin{bmatrix} 0.8078 & -0.0081 \\ 1.6579 & 0.8078 \end{bmatrix}}_A \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \underbrace{\begin{bmatrix} 0.0969 \\ 0.0789 \end{bmatrix}}_B e[n] \\ u[n] &= \underbrace{\begin{bmatrix} -0.0611 & -0.0065 \end{bmatrix}}_C \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \underbrace{0.0518}_D e[n] \end{aligned}$$

Paso 4: Control Anti-windup

Para el cálculo de la ganancia del control anti-windup L usamos la función `kalman(.)` en Julia o MATLAB. Realizando este cálculo, obtenemos el siguiente vector de ganancia:

$$L = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} = \begin{bmatrix} 1.8477 \\ 1.3698 \end{bmatrix}.$$

Paso 5: Implementación del algoritmo de control

Para implementar el algoritmo de control, encontramos primero las matrices $A_{aw} = A - LC$ y $B_{aw} = B - LD$. Con los valores numéricos para A_{aw} y B_{aw} , las ecuaciones (1.10) y (1.11) para el controlador que estamos implementando toman la siguiente forma:

$$\begin{aligned} \begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix} &= \underbrace{\begin{bmatrix} 0.9219 & 0.0041 \\ 1.7499 & 0.8177 \end{bmatrix}}_{A_{aw}} \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \underbrace{\begin{bmatrix} 0.0001223 \\ 0.0008439 \end{bmatrix}}_{B_{aw}} e[n] + \underbrace{\begin{bmatrix} 1.8477 \\ 1.3698 \end{bmatrix}}_L u[n] \\ u[n] &= \text{sat} \left(\underbrace{\begin{bmatrix} -0.0611 & -0.0065 \end{bmatrix}}_C \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \underbrace{0.0518}_D e[n] \right) \end{aligned}$$

1.3.1 Pseudocódigo

```

1 // Main control algorithm - critical repeating task
2 for (;;) {
3     r = read_reference()      // read setpoint
4     y = read_sensor()        // read process variable
5     u = computeController(e, umin, umax);
6     //compensateDeadZone(u) if necessary
7     usat = sat(u, umin, umax);
8     sendAnalogOutput(usat);
9
10    ...Repeat this task when the sampling time has elapsed.
11 }
```

Listing 1.1: Rutina principal de control

El listado 1.1 presenta un esquema básico de algoritmo de control. Notemos que en la linea 6 se usa la función `u = computeController(.)`, cuya función es calcular, en cada ciclo de muestreo, la señal de control que debe aplicarse al sistema. El código de esta función, generado automáticamente por el *notebook* adjunto a este documento, se muestra en el listado 1.2.

```

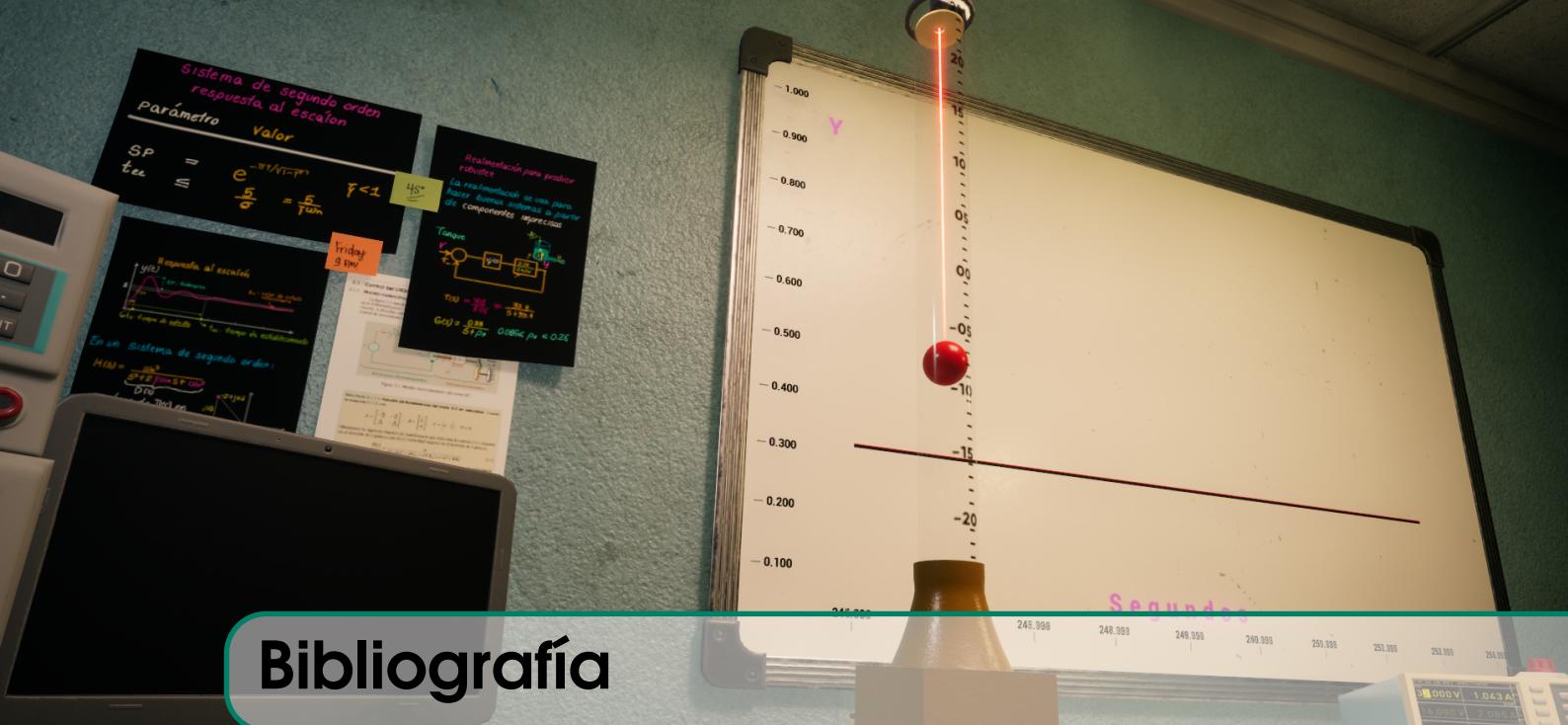
1 float computeController(float e, float umin, float umax){
2     // This function computes the control law for a discretized controller Cd(z)
3     // e = ref - y is the current error in the controlled system
4     // and umin and umax are the limits of the control signal
5
6     // The following constants define the Ab matrix
7     const float aw11 = 0.92190784215927124023;
8     const float aw12 = 0.00414303503930568695;
9     const float aw21 = 1.74990832805633544922;
10    const float aw22 = 0.81766259670257568359;
11
12    // The following constants define the Bb matrix
13    const float bw1 = 0.00012232111475896090;
14    const float bw2 = 0.00084393681026995182;
15
16    // The following constants define the C matrix
17    const float c1 = -0.06108257919549942017;
18    const float c2 = -0.00654000043869018555;
19
20    // The following constant define the D scalar
21    const float d1 = 0.05182243138551712036;
22
23    // The following constants define the antwindup vector L
24    const float l1 = 1.86796653270721435547;
25    const float l2 = 1.50687634944915771484;
26
27    // The following variables represent the states x[n]
28    // in the state-space representation. They must be declared
29    // as static to retain their values between function calls.
30    static float x1 = 0;
31    static float x2 = 0;
32
33    // The following variables are the new computed states x[n+1]
34    // of the state space representation
35    float x1_new = 0;
36    float x2_new = 0;
37
38    // The following variable is the control signal.
39    // it also must be declared as static to retain its value between function calls.
40    static float u = 0;
41
42    ****
43    THIS IS THE CONTROLLER'S CODE
44    ****
45
46    // Compute the new predicted state x[n+1] = Ab*x[n] + Bb*e[n] + L*u[n]
47    x1_new = aw11*x1 + aw12*x2 + bw1*e + l1*u;
48    x2_new = aw21*x1 + aw22*x2 + bw2*e + l2*u;
49
50    // Compute the control output u[n] = C*x[n] + D*e[n]
51    u = c1*x1 + c2*x2 + d1*e;
52
53    // Saturate control signal
54    u = constrain(u, umin, umax);
55
56    // Make the predicted state the current state: x[n] <- x[n+1]
57    x1 = x1_new;
58    x2 = x2_new;
59
60    // now, the control signal is available to the main control routine
61    return u;

```

62 }

Listing 1.2: Rutina principal de control

La mayor parte del código es la definición de los parámetros y variables del controlador. El algoritmo de control, como tal, se escribe en solo 6 líneas de código que aparecen entre las líneas 42–61 del listado 1.2.



Bibliografía

Libros

- [ÅW97] Karl J. Åström y Björn Wittenmark. *Computer-controlled systems (3rd ed.)* USA: Prentice-Hall, Inc., 1997. ISBN: 0133148998 (véanse páginas 4, 7).
- [LE19] B.J. Lurie y P. Enright. *Classical Feedback Control with Nonlinear Multi-Loop Systems: With MATLAB and Simulink, Third Edition*. Automation and Control Engineering. CRC Press, 2019. ISBN: 9781351011839. URL: <https://books.google.com.co/books?id=xienDwAAQBAJ> (véase página 3).

Artículos

Online

